QBUS6850

Lecture 4

Scalable Classification Methods

BUSINESS SCHOOL

Professor Junbin Gao

THE UNIVERSITY OF
SYDNEY

❑ Topics covered

- k-Nearest-Neighbours Algorithm
- Unsupervised learning introduction
- K-means

❑ References

- Friedman et al., (2001), Chapters 13.1 - 13.3, 14.3
- James et al., (2014), Chapter 10.3
- Bishop, (2006), Chapter 2.5.2, 9.1

Learning Objectives

- □ Understand the intuition of k-NN
- □ Understand how k-NN works
- □ Understand the mode specification of k-NN
- □ Understand the classification confusion matrix
- □ Understand precision and recall, F1-score, ROC etc
- □ Understand the intuition of K-means
- □ Understand the loss function of K-means and how it works
- □ Understand why and how to do random initialization for K-means

k-Nearest-Neighbours Algorithm

k-NN Classifier

k-Nearest-Neighbours is a classification algorithm that in order to determine the classification of a point, and combines the classification of the k nearest points.

k-NN requires no distribution assumptions.

It is a **supervised learning algorithm**: classify a point based on the known classification of other points.

k-NN Classifier

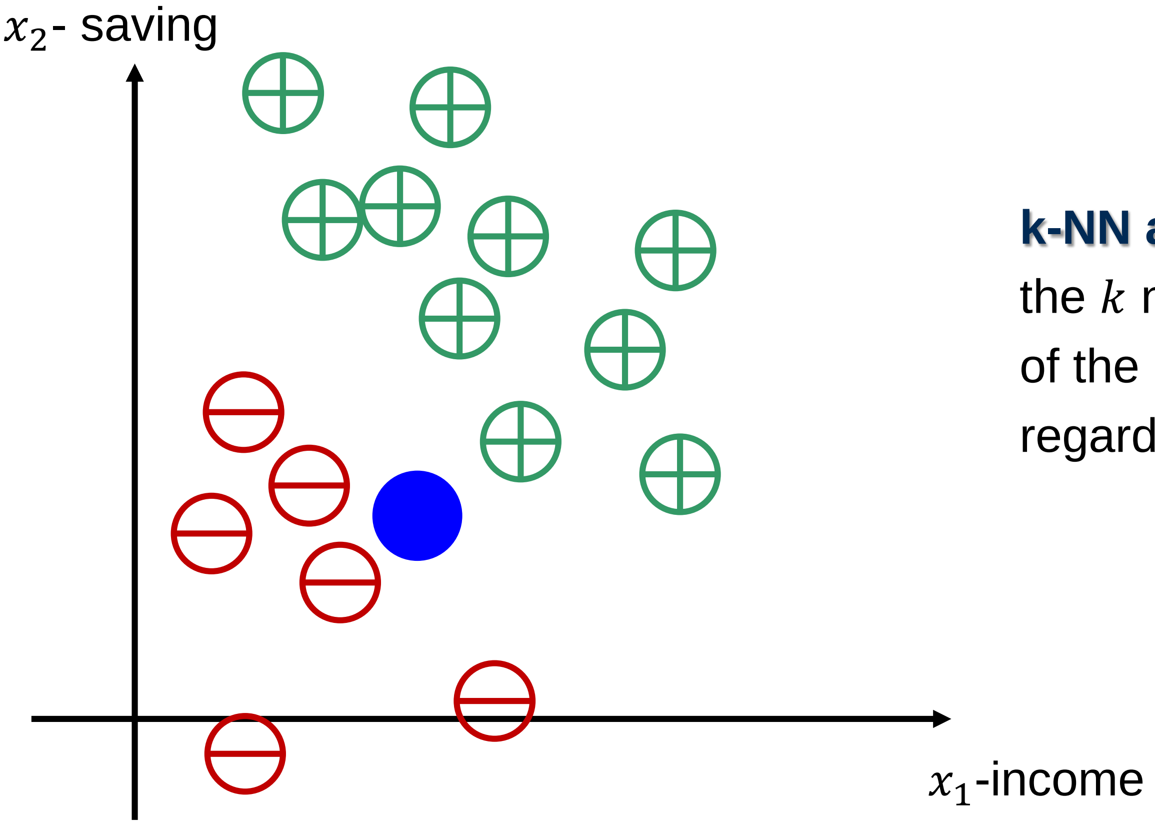
How it works?

❑ Find k nearest neighbours of x.

❑ Predict the class of x to be the majority class of the k nearest neighbours.

❑ Parameter k can be determined using validation.

k-NN Intuition

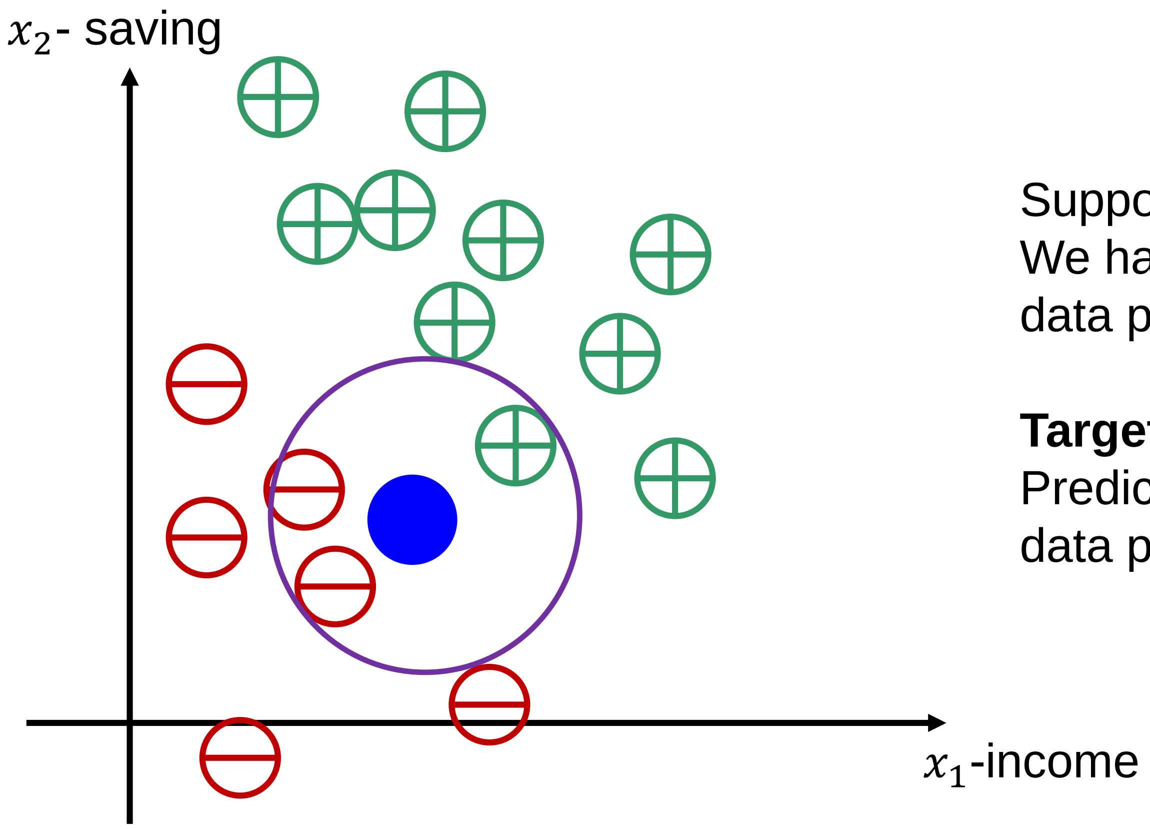


k-NN algorithm identifies the k nearest neighbours of the blue data point, regardless of the label

Training set: $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \dots, (\mathbf{x}_N, t_N)\}$

where each $\mathbf{x}_n = (x_{n1}, x_{n2}) = (\text{income}, \text{saving})$ and t_n is ⊕ or ⊖

k-NN Intuition

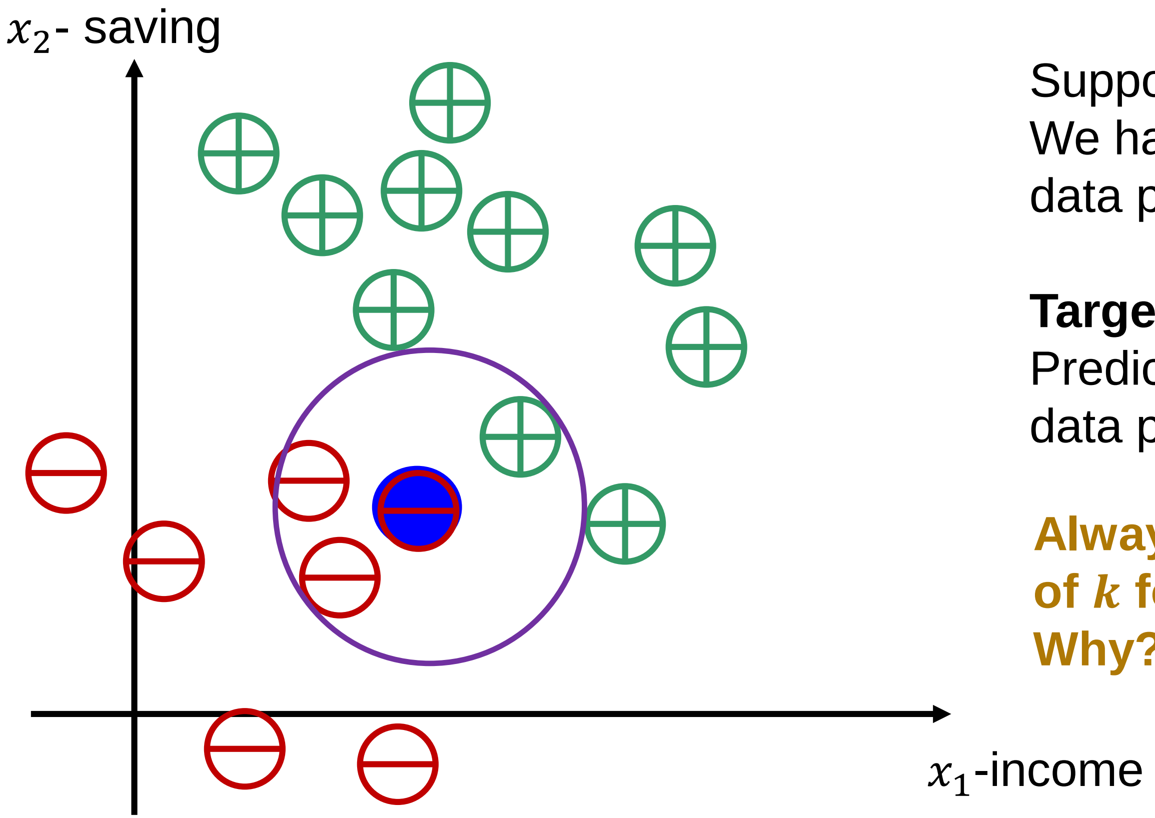


Suppose: $k = 3$
We have red and green labelled data points

Target:
Predict the class of the blue data point.

Training Set: $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \dots, (\mathbf{x}_N, t_N)\}$

k-NN Intuition



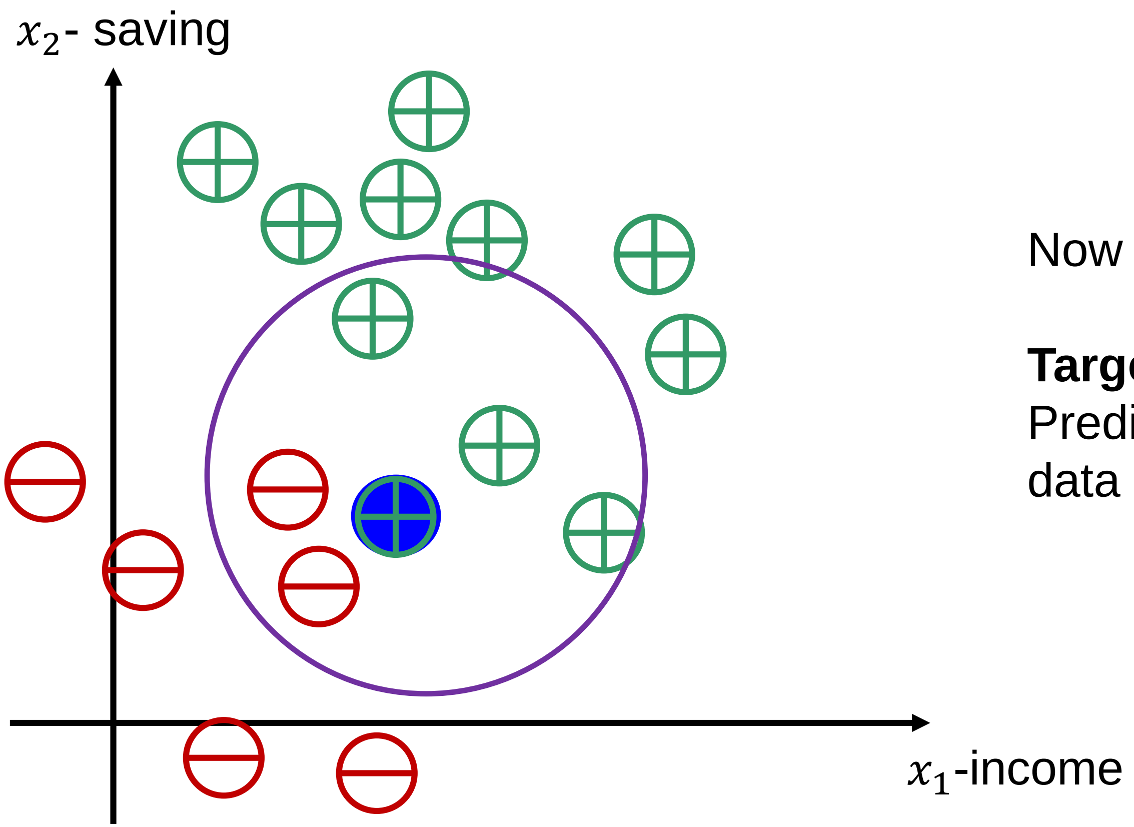
Suppose: $k = 3$
We have red and green labelled data points

Target:
Predict the class of the blue data point: red class

Always choose ODD values of k for a 2 class problem. Why?

Training Set: $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \dots, (\mathbf{x}_N, t_N)\}$

k-NN Intuition



Now $\boldsymbol{k} = \mathbf{5}$

Target:
Predict the class of the blue data point: green class

Training Set: $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \dots, (\mathbf{x}_N, t_N)\}$

k-NN Model

This is a nonparametric model. There is no need for parameter estimation.

Training Set: $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \ldots, (\mathbf{x}_N, t_N)\}$

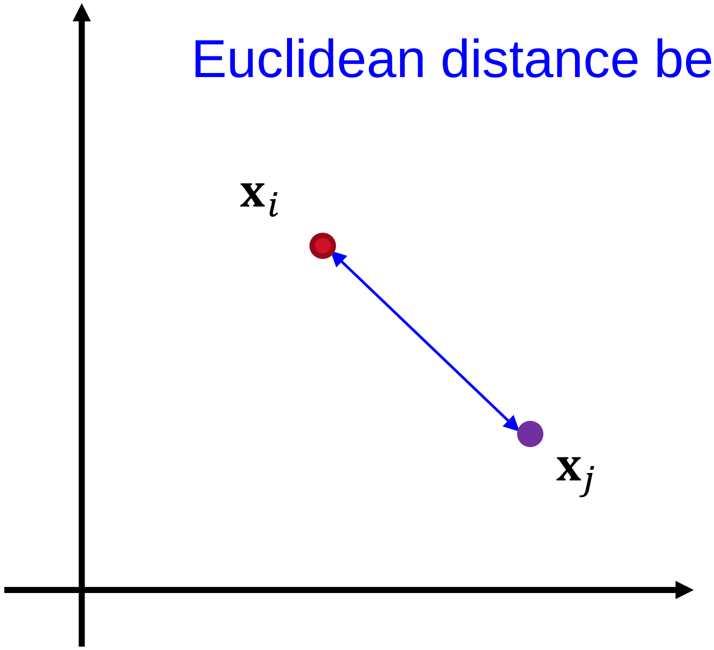
We need an algorithm for this

The model is defined as

$$f(\mathbf{x}, \beta) = \text{Mode}\{\mathbf{x}_n \in N_k(\mathbf{x}) \text{ where } \mathbf{x}_n \in \mathcal{D}\}$$

where $N_k(\mathbf{x})$ is the neighbourhood of $\mathbf{x}$ defined by ***k* closest points** in the training dataset, and Mode() is the Mode function (regarding t_n values of $\boldsymbol{k}$ closest points).

k Closest Points


Euclidean distance between $\mathbf{x}_i$and $\mathbf{x}_j$.
$\mathbf{x}_i$
$\mathbf{x}_j$

$$\|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)}$$

There are other distance measures that can be used.

The k-NN prediction rule for regression:
For a new point $\mathbf{x}_0$, the model predicts the value at $\mathbf{x}_0$ as the average value of all the values at the $\boldsymbol{k}$ neighbours in $N_k(\mathbf{x}_0)$

The k-NN prediction rule for classification:
For a new point $\mathbf{x}_0$, the model predicts the class of $\mathbf{x}_0$ as the majority vote (mode) among the $\boldsymbol{k}$ neighbours in $N_k(\mathbf{x}_0)$.

k = 1, 1-Nearest-Neighbor

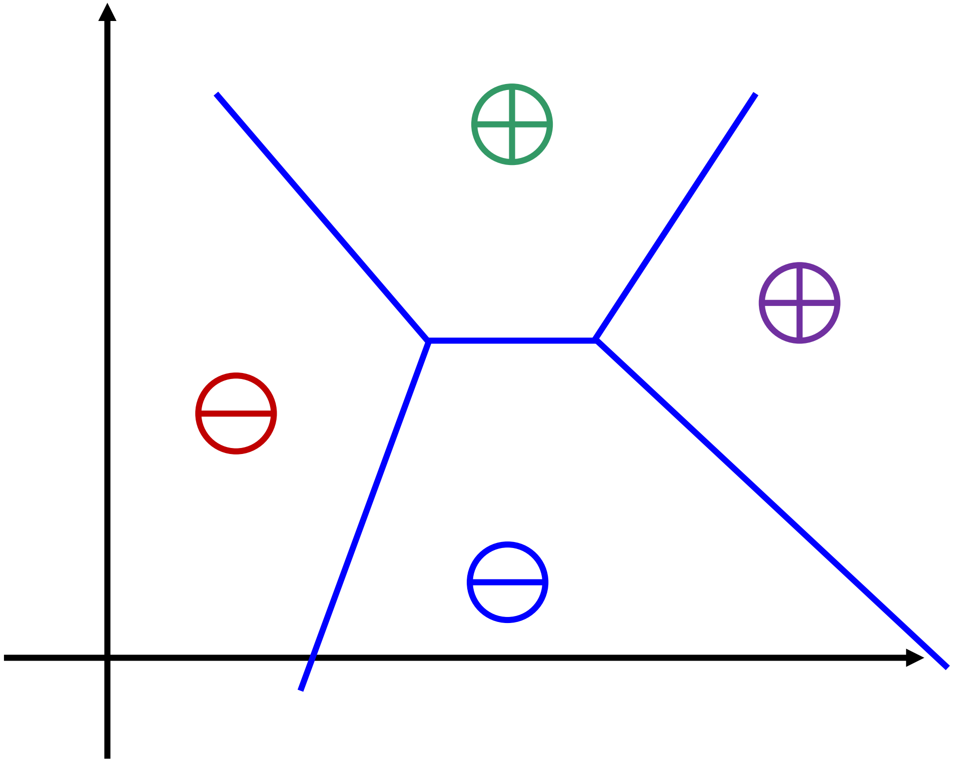
❑ **x** is one of training data, then who are the neighbourhood? $\mathbf{x}$ is its own neighbourhood.

❑ If $k = 1$, there is no misclassification for the training data

❑ 1-NN is more accurate on the training data, while causing overfitting problem. The algorithm is also quite sensitive to outliers.

❑ “kernel” methods (later in the course) use a generalized notion of neighbourhood but are in essence same thing as k-NN.

k = 1, 1-Nearest-Neighbor



Each training vector defines a region in space, resulting a Voronoi partition of the space

Georgy Voronoy (1868-1908)

k=?

Always choose ODD values of *k* for a 2 class problem.

k cannot be a multiple of the number of class, to avoid a tie when assigning class.

Loss function for classification/Confusion matrix

k-NN: Model Selection

The misclassification rate is: FP (false positive) and FN (false negative)

$$\frac{\mathrm{FP} + \mathrm{FN}}{\text{Total \# of predictions}}$$

- The target of k-NN?
- Minimize number of misclassification of training set? This can be minimized at *k* = 1. Might not the good way.
- There is a *k* for which bias and variance are balanced optimally.
- How to decide *k*?
- We can still incorporate the model and features techniques, including training & validation & test sets, CV, etc, to see which *k* is the best regarding the target metrics, e.g. **misclassification rate.**

- **Is misclassification rate always a good metric for all applications?**

Skewed Data

- In the customer default example, we predict t =1 if default, and t = 0 if not default.
- Suppose using logistic regression or k-NN algorithm, we get 99% accuracy rate, or 1% misclassification rate (error rate).
- In reality, there is approximately 0.5% customers would default (t = 1).

Is this a good classifier / machine learning model?

- What if we build a simpler classifier that predicts everyone to be t = 0 (not default)?
- This classifier will only have 0.5% misclassification rate.
- Is it a better classifier?

Skewed data!

Confusion Matrix

		Predicted values	
		0 (-)	1 (+)
Actual value	0 (-)	**True negative (TN)**	**False positive (FP)**
	1 (+)	**False negative (FN)**	**True positive (TP)**

t =1 represents the rare class that we want to capture, e.g. default

The recall, or sensitivity rate, or true positive rate (TPR) is:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = \frac{\mathrm{TP}}{\boxed{\text{Total \# of ACTUAL Positives}}}$$

Second ROW sum of the confusion matrix

For **all customers who actually** default, what is the percentage that we correctly predicted as *t* =1 (default). Higher the better.

The precision (positive predictive rate) is:

$$\mathrm{PPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = \frac{\mathrm{TP}}{\boxed{\text{Total \# of PREDICTED Positives}}}$$

Second COLUMN sum of the confusion matrix

For **all customers we predict** *t* =1 (default), what is the percentage of them who actually default. Higher the better.

Why not build a simpler classifier that predict everyone to be *t* = 0 (not default)?

The recall will be 0=> to avoid cheating

We check both precision and recall instead of just evaluating one.

Confusion Matrix

- ❑ Recall — how good a test is at detecting the positives. A test can cheat and maximize this by always returning “positive”.

- ❑ Precision – how many of the positively classified were relevant. A test can cheat and maximize this by only returning positive on one result it is most confident.

- ❑ The cheating is resolved by looking at both metrics instead of just one.

t actual	0	0	0	0	0	1	0	1	0	0
t predicted-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
t predicted-2	0	0	0	0	0	1	0	0	0	0

Calculate the precision and recall for each model.

Precision and Recall Tradeoff

t actual	0	0	0	0	0	1	0	1	0	1
t predicted-A	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
t predicted-B	0	0	0	0	0	0	0	**1**	0	0

For model A: we predict t =1 even if we are NOT that confident, e.g. low probability threshold of logistic regression

Recall=3/3= 100%

Precision=3/9=33.33%

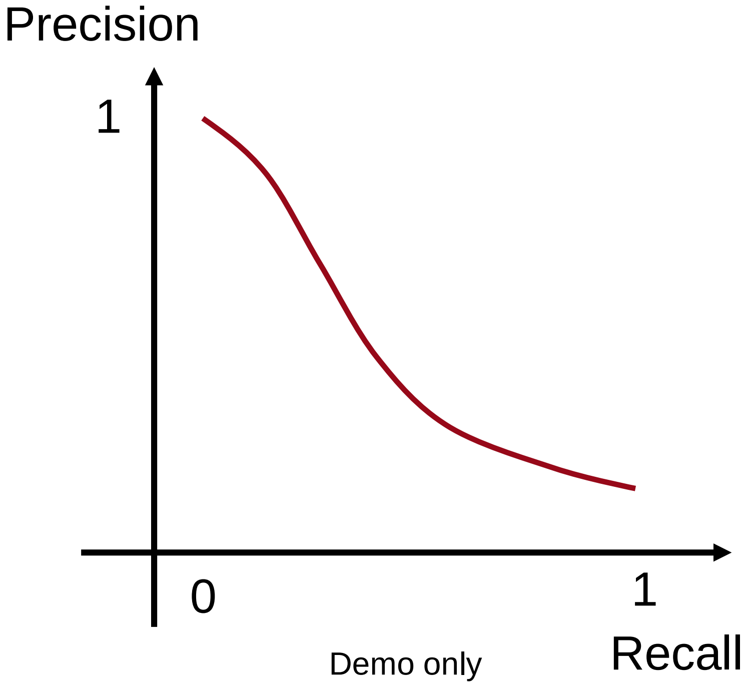
High recall, low precision.

For model B: we only predict t =1 only if we are VERY confident, e.g. high probability threshold of logistic regression

Recall= 1/3= 33.33%.

Precision=1/1=100%

Low recall, high precision,.



THE UNIVERSITY OF SYDNEY

F1-Score

$$\mathrm{F1-Score} = \boxed{2\frac{1}{\frac{1}{\mathrm{Precision}}+\frac{1}{\mathrm{Recall}}}} = 2\frac{\boxed{\mathrm{Precision} * \mathrm{Recall}}}{\mathrm{Precision} + \mathrm{Recall}}$$

Harmonic mean

If a classifier is cheating, then precision or recall are likely to be close to 0, then F1-Score will be close to 0.

To evaluate a classifier:

- **Incorporate the model and feature selection techniques, including training & validation & test sets and CV**
- **Evaluate misclassification rate, precision & recall and F1-Score.**

Other Measures for Binary Classifiers

- The confusion matrix for a binary classifier again

		Predicted values	
		0 (-)	1 (+)
Actual values	0 (-)	**True negative (TN)**	**False positive (FP)**
	1 (+)	**False negative (FN)**	**True positive (TP)**

- Define the true positive rate (TPR) and the false positive rate (FPR)

$$TPR = \frac{TP}{TP + FN} \quad \text{and} \quad FPR = \frac{FP}{FP + TN}$$

- Both jointly assess the performance of a classifier. The best case would be a higher TPR and a lower FPR. This means the classifier wont make many errors on positive and negative examples, respectively

Receiver Operating Characteristic (ROC)

- Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality. Particularly useful for binary classification classifiers
- ROC curves typically feature true positive rate (TPR) on the Y axis, and false positive rate on (FPR) the X axis. This means that the top left corner of the plot is the “ideal” point — a false positive rate of zero, and a true positive rate of one.
- This is not very realistic, but it does mean that a larger Area Under the Curve (AUC) is usually better.
- ROC curves are typically used in binary classification to study the output of a classifier. In order to extend ROC curve and ROC area to multi-class or multi-label classification, it is necessary to binarize the output. One ROC curve can be drawn per label.

Receiver Operating Characteristic (ROC)

- ❑ Take logistic regression classifier as an example
- ❑ For all the test/validation examples, the logistic regression model produces the probability for each case/example to be class A (or Class 1).
- ❑ A typical decision is if the probability (or score) is larger than 0.5, then we classify the case as Class A (or Class 1), otherwise Class B (or Class 0).
- ❑ Why do we use 0.5? In fact, taking a threshold between 0.0 and 1.0, we can make decision based on the same training model.
- ❑ For each threshold between 0.0 and 1.0, we can have decisions over all the examples, then we can construct a confusion matrix, then we can calculate TPR and FPR, and finally draw a point on the coordinates.

ROC: Example

❑ In test dataset:100 positive cases (Class 1) and 100 negative cases (Class 0)

❑ Take threshold as 0.0, then all the testing cases will be classified as positive. Why? Hence TPR=1.0 and FPR=1.0

❑ Take the threshold as 1.0, when all the testing cases will be classified as negative. Why? Hence TPR=0.0 and FPR=0.0

❑ Threshold = 0.25 produces TPR=0.9 and FPR=0.60

Threshoold =0.25		**Predicted**	
		0 (-)	1 (+)
True Test	0 (-)	**40**	**60**
	1 (+)	**10**	**90**

ROC: Example

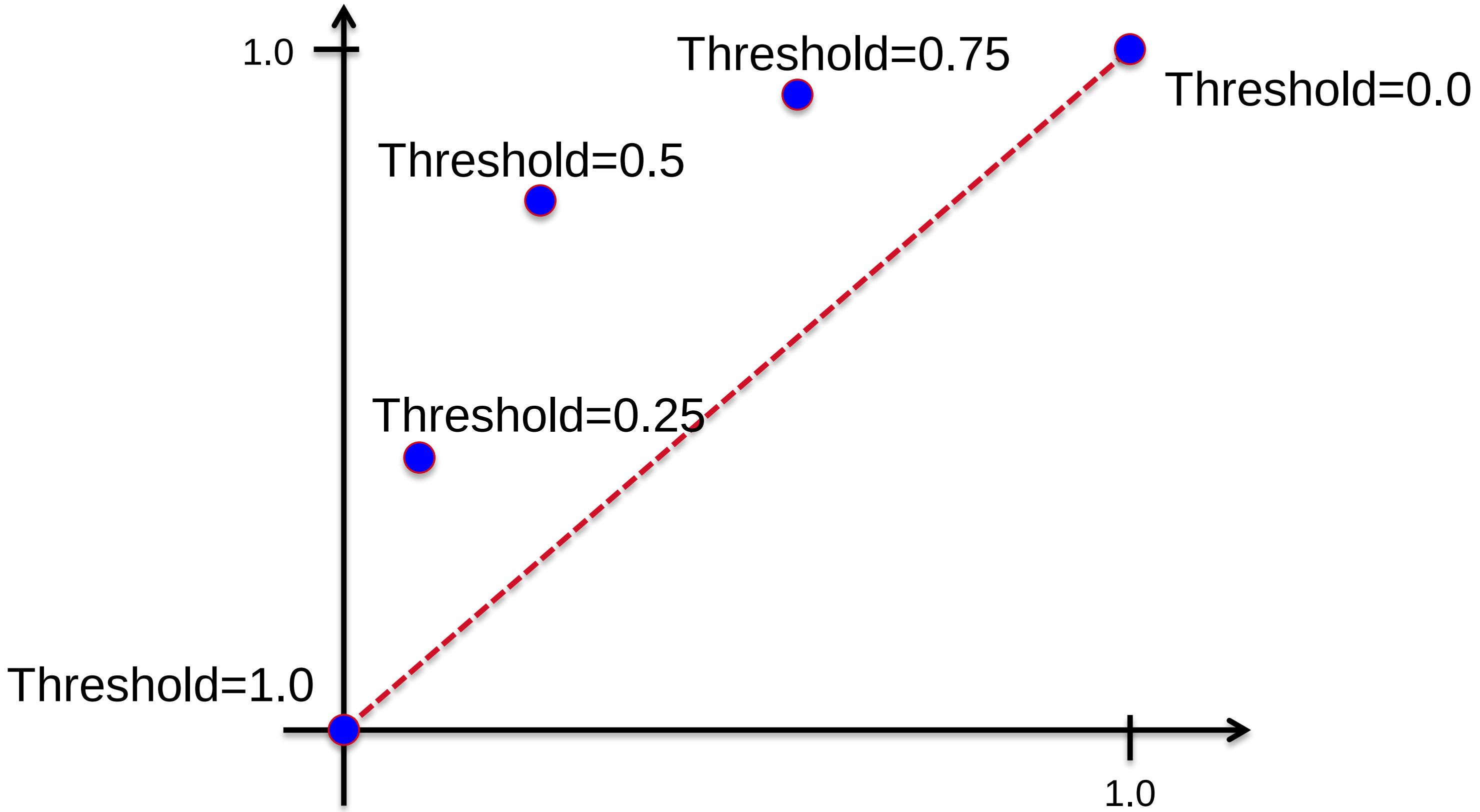
❑ Threshold = 0.50 produces TPR=0.80 and FPR=0.20

Threshoold d =0.50		**Predicted**	
		0 (-)	1 (+)
True Test	0 (-)	**80**	**20**
	1 (+)	**20**	**80**

❑ Threshold = 0.75 produces TPR=0.40 and FPR=0.10

Threshoold d =0.70		**Predicted**	
		0 (-)	1 (+)
True Test	0 (-)	**90**	**10**
	1 (+)	**60**	**40**

ROC: Example



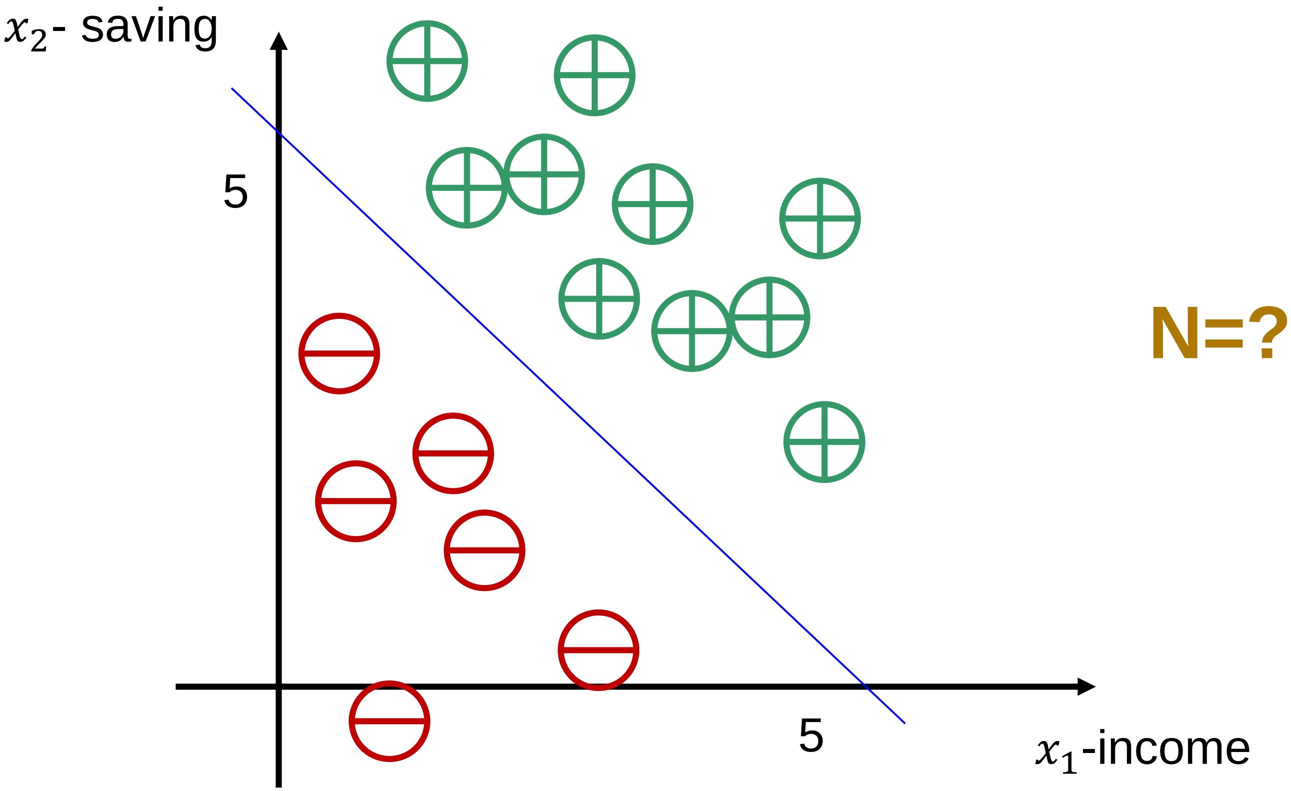
❑ This finally gives us the ROC. sklearn can automate the process of finding ROC. See `Lecture04_Example00.py`

Python Example

(Lecture04_Example01.py)

Unsupervised Learning

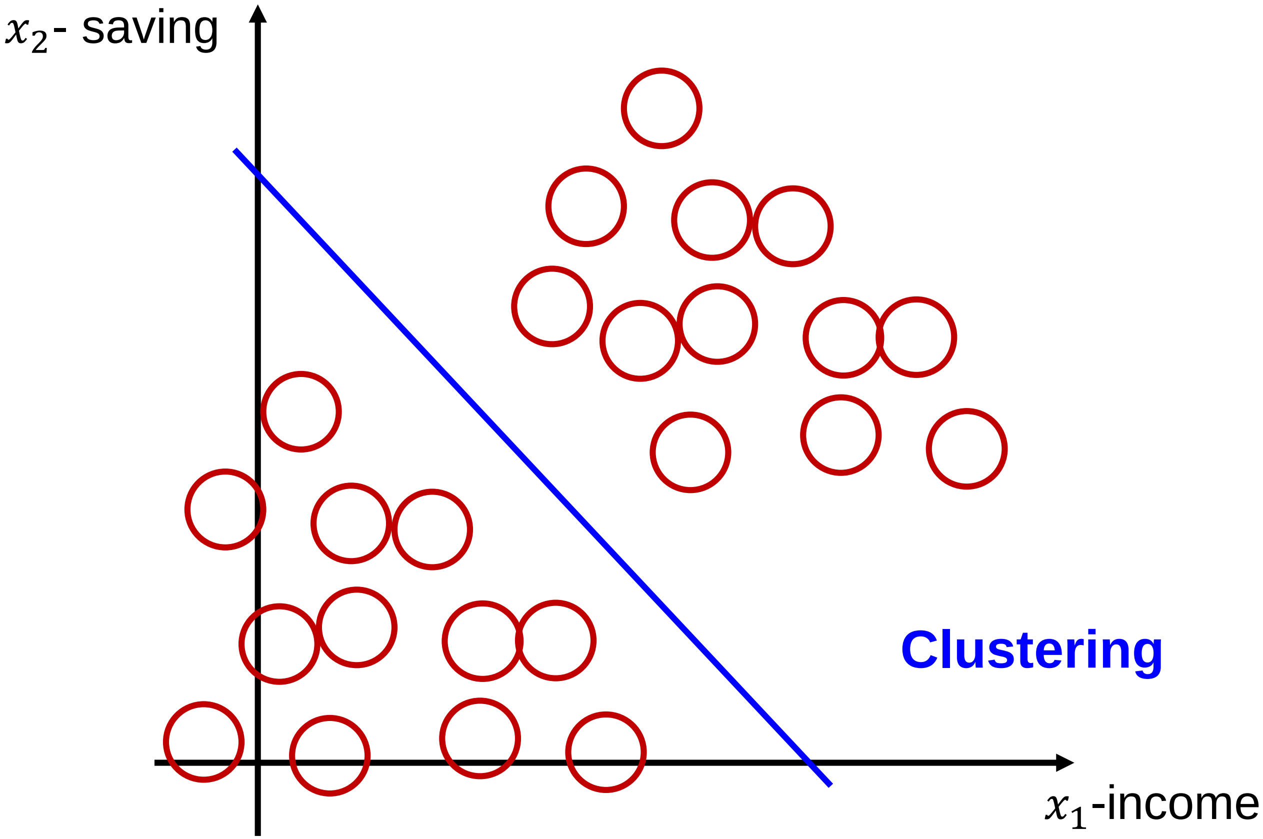
Supervised Learning



N=?

Training Set: $\mathcal{D} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), \dots, (\mathbf{x}_N, t_N)\}$

Unsupervised Learning



d=?
Number of features=?

In unsupervised learning we do not make the distinction between the response/target variable (*t*) and the feature (**x**).

Training Set: $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_N\}$

Applications of Clustering

❑ Customer segmentation

❑ Social network analysis

❑ Market segmentation

❑ Image clustering

K-means Algorithm

K-means

K-means is a clustering algorithm that tries to partition a set of points into K sets (clusters) such that the points in each cluster tend to be near each other.

It is a **unsupervised learning algorithm**, since data are not labelled**.**

K-means Input

K (number of clusters)

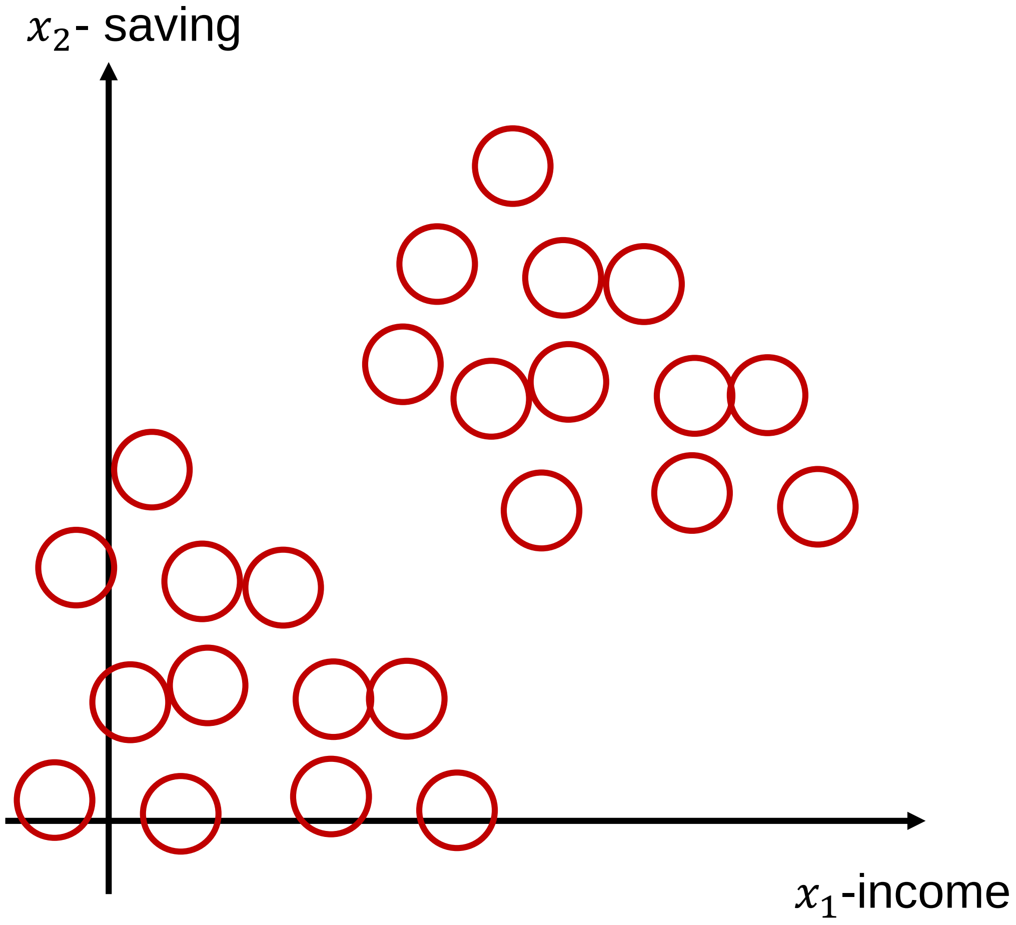
Training Set: $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_N\}$

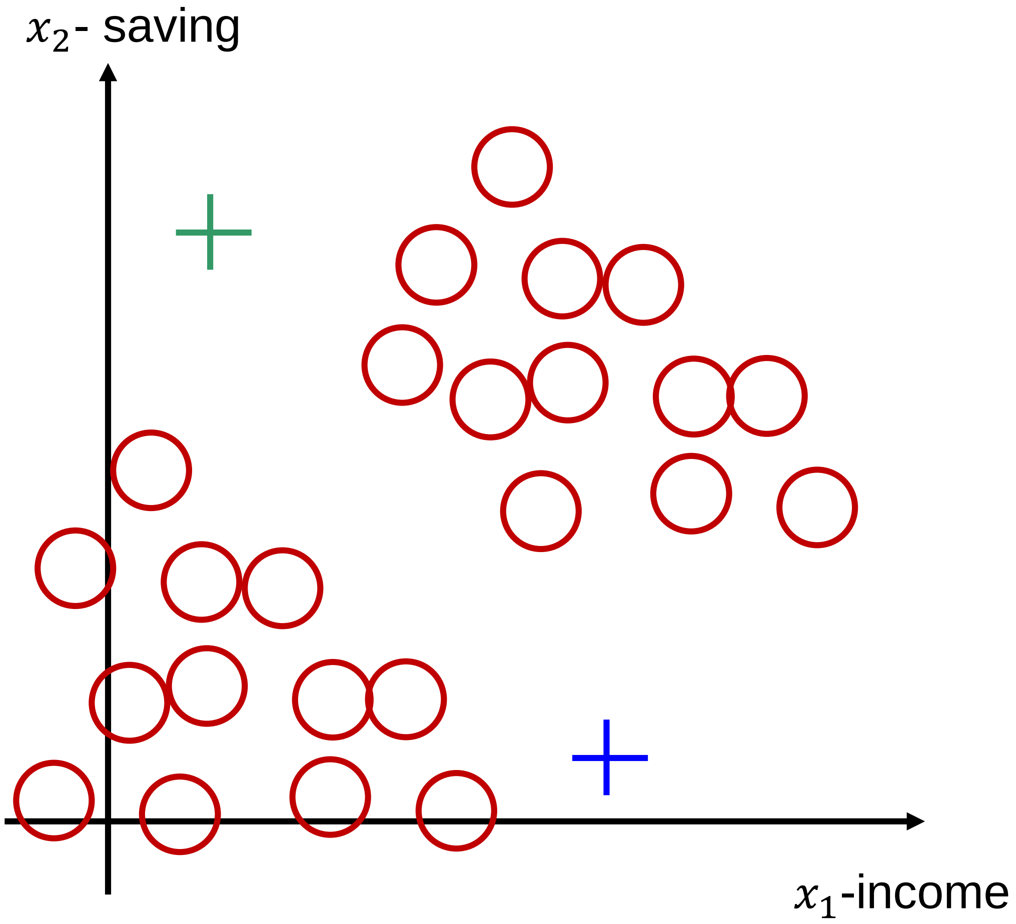
How to choose *K*?
How to select the starting values of cluster centroids?

K-Means is an iterative algorithm and it contains **two** steps.

i. cluster assignment step
ii. centroid move step.

How K-means work



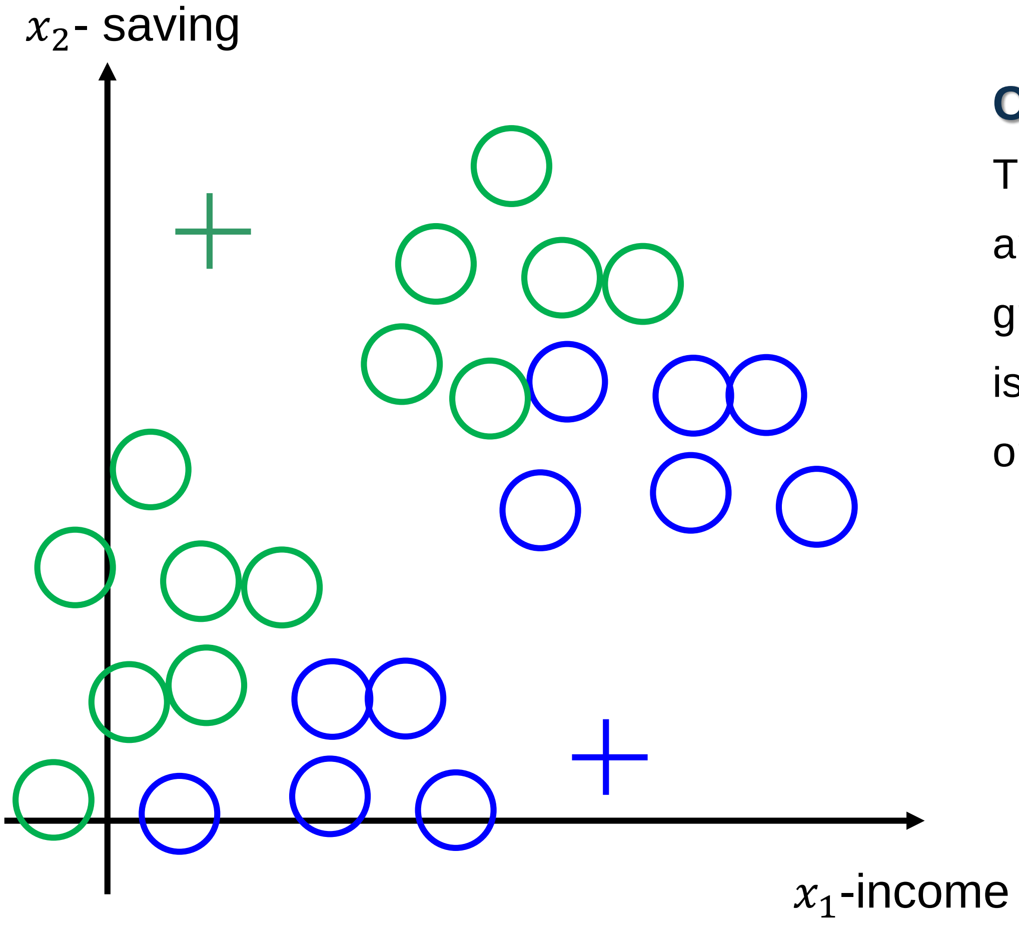


Loop 1

Randomly pick **two** (why?) clusters centroids;

The first of the two steps in the loop of K-means: **cluster assignment step**.

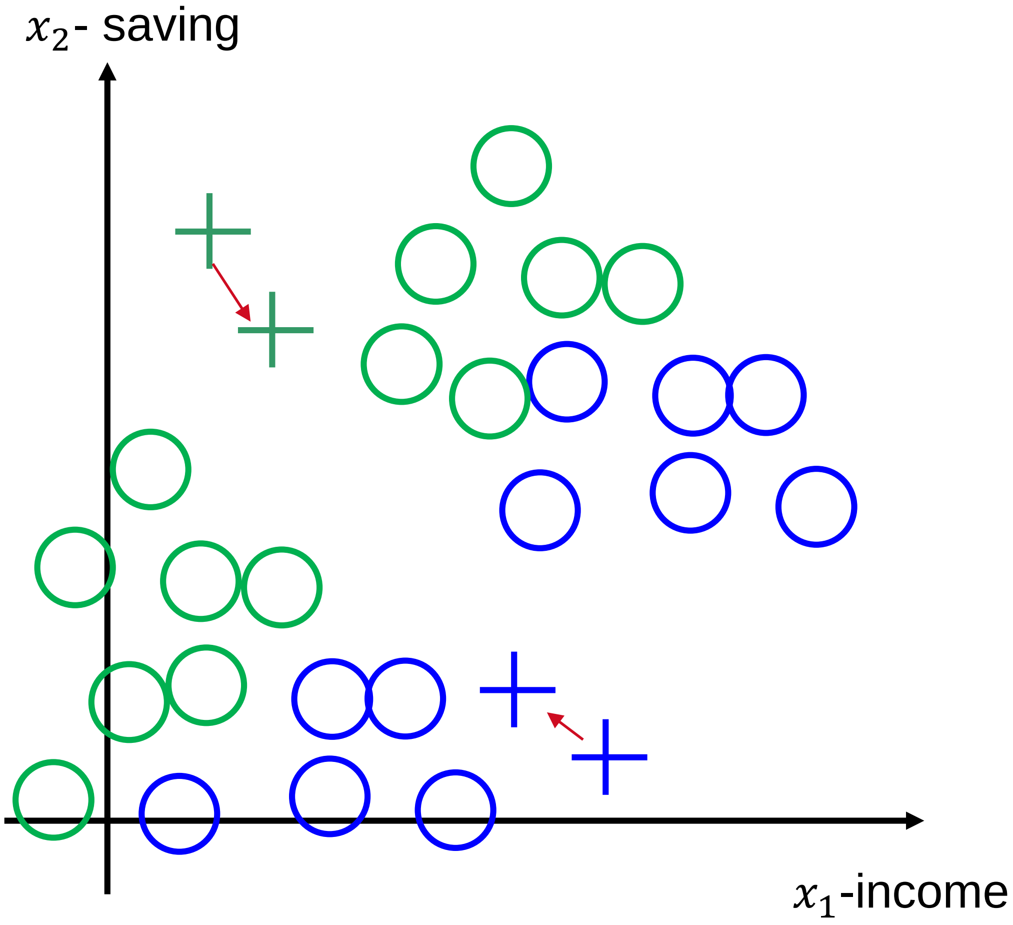
Go through each of the red observation, and decide each of them is closer to the green cluster centroid or the blue cluster centroid.



Loop 1

Cluster Assignment Step.

This step is to go through the data set and color each of the points either green or blue, depending on whether it is closer to the green cluster centroid or the blue cluster centroid.



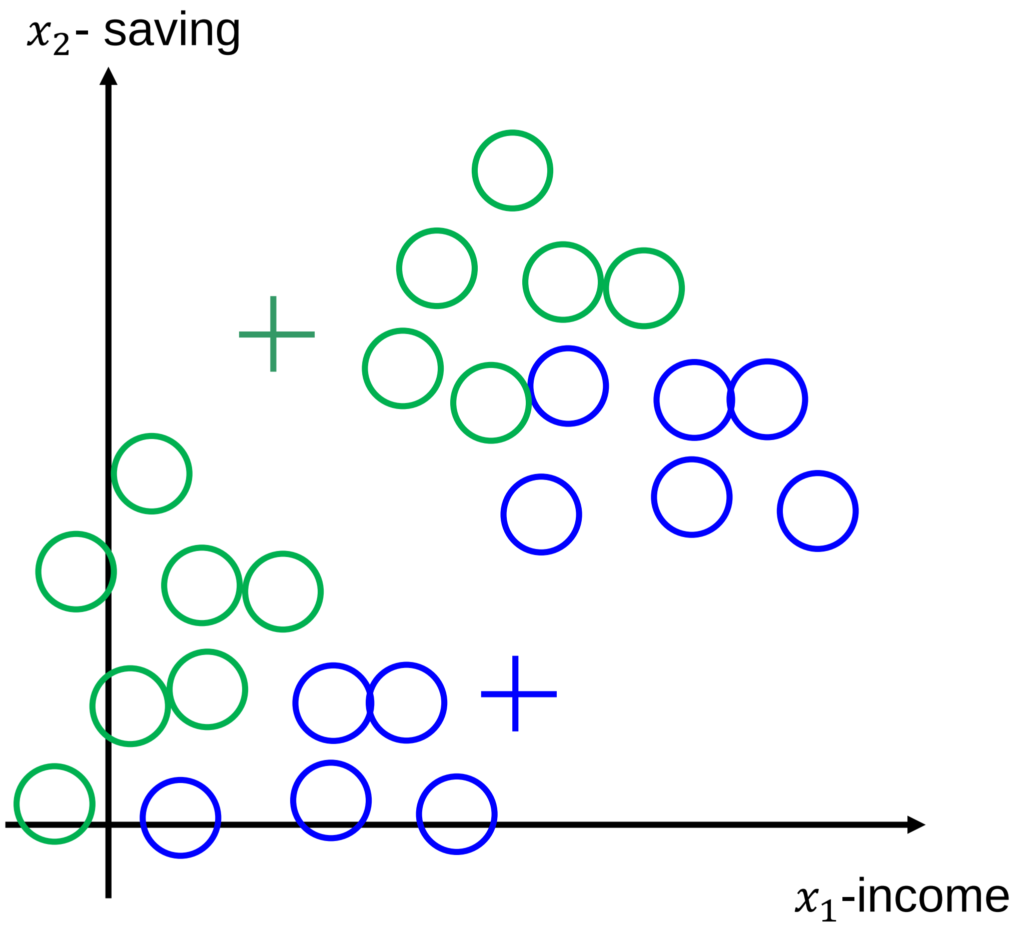
Loop 1

Centroid move step.

Take the two cluster centroids (green and blue crosses), and move them to the average of the points colored with the same colour.

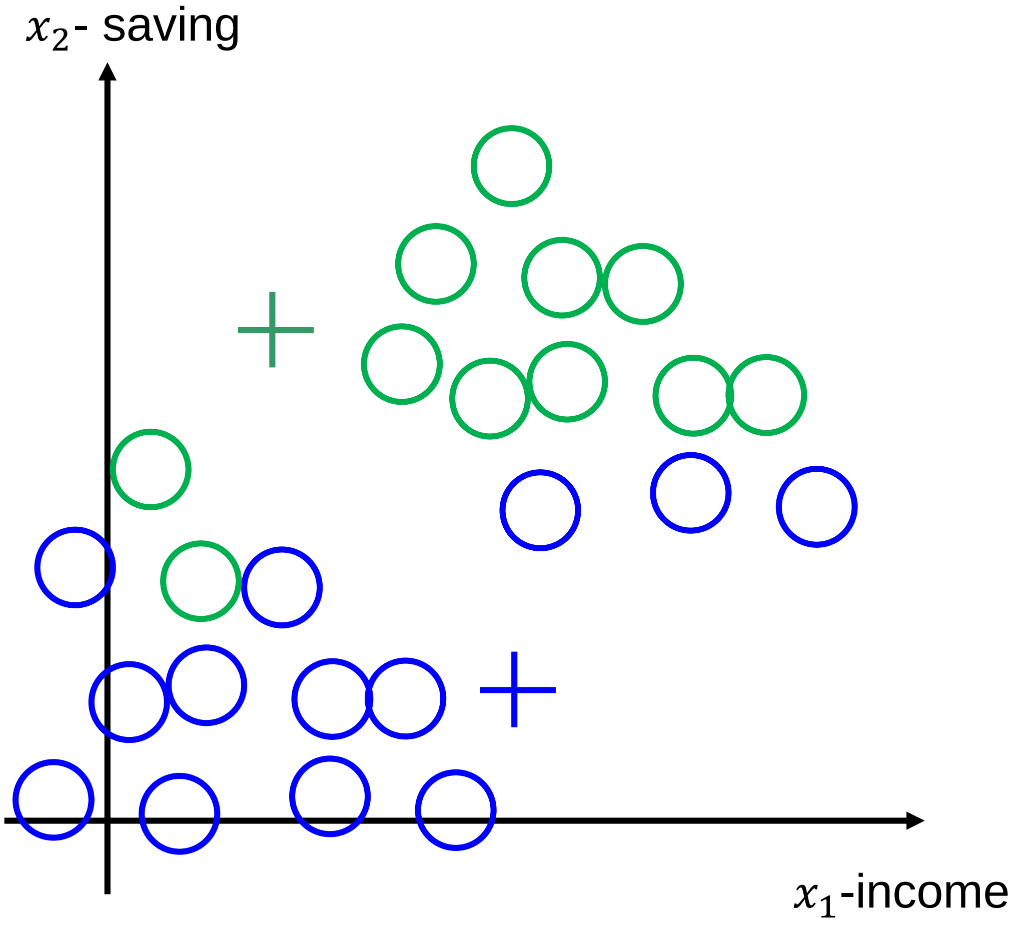
Specifically:

Check all the green points and compute the average, and move the green cluster centroid to the average. Do this for the blue points and centroid as well.



Loop 1

New Centroids' locations.

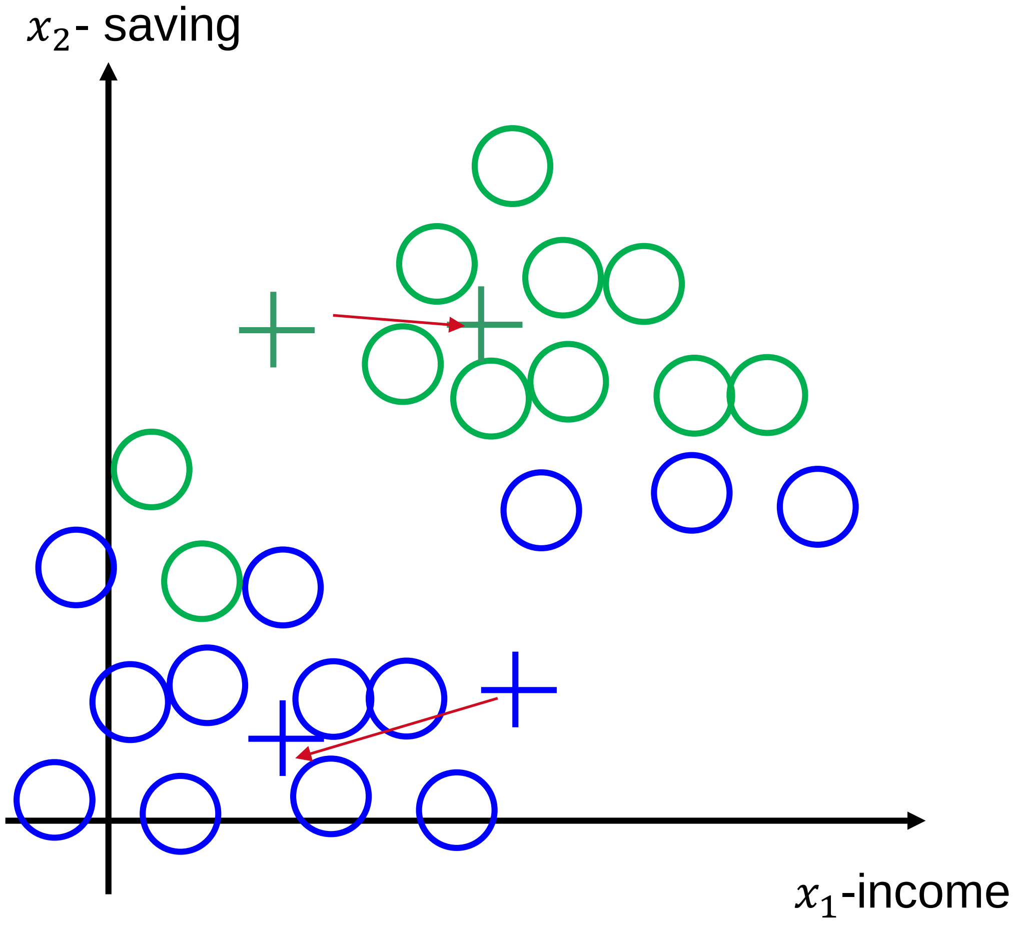


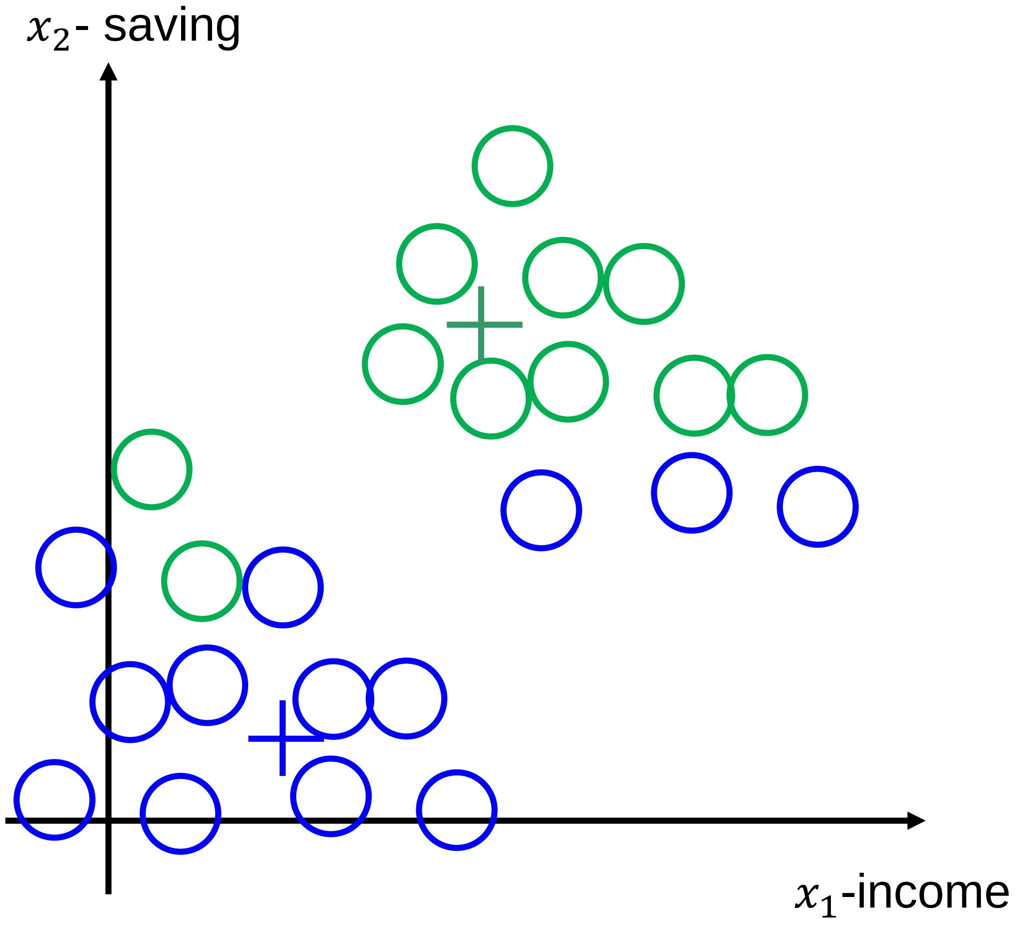
Loop 2

Cluster Assignment Step.

Loop 2

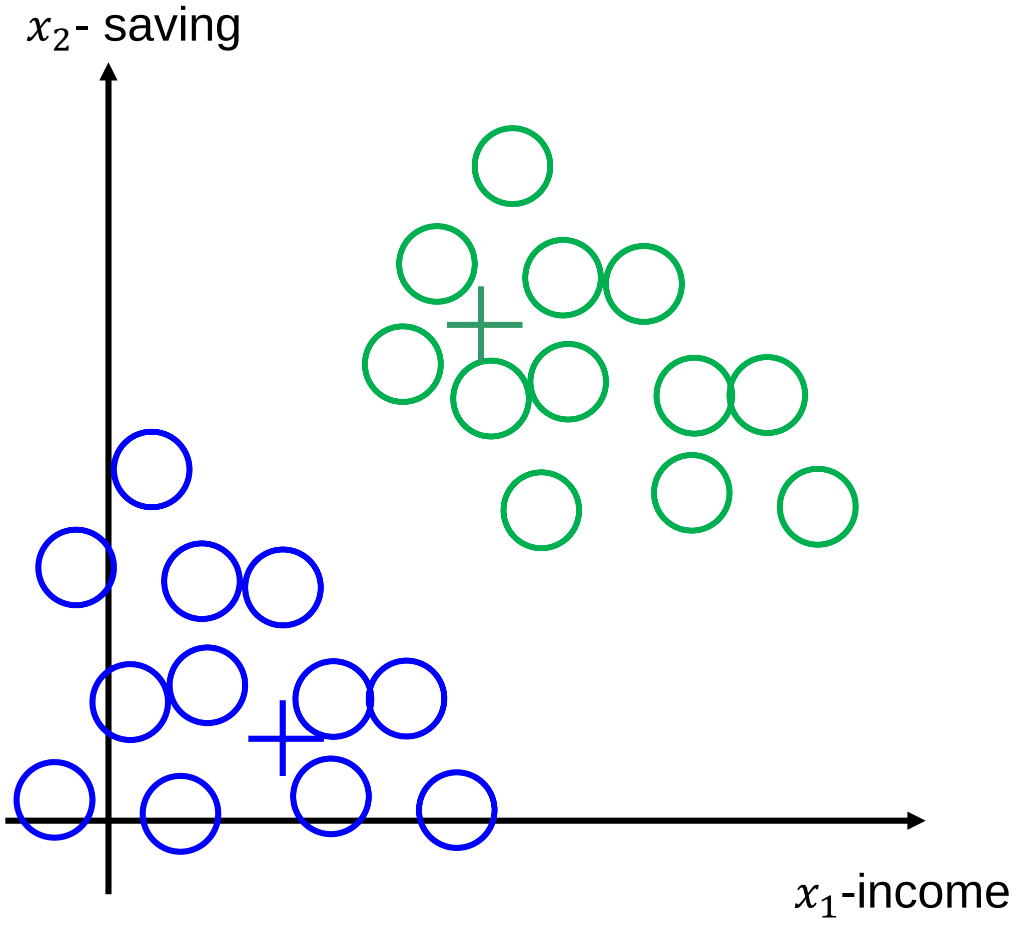
Centroid Move Step.





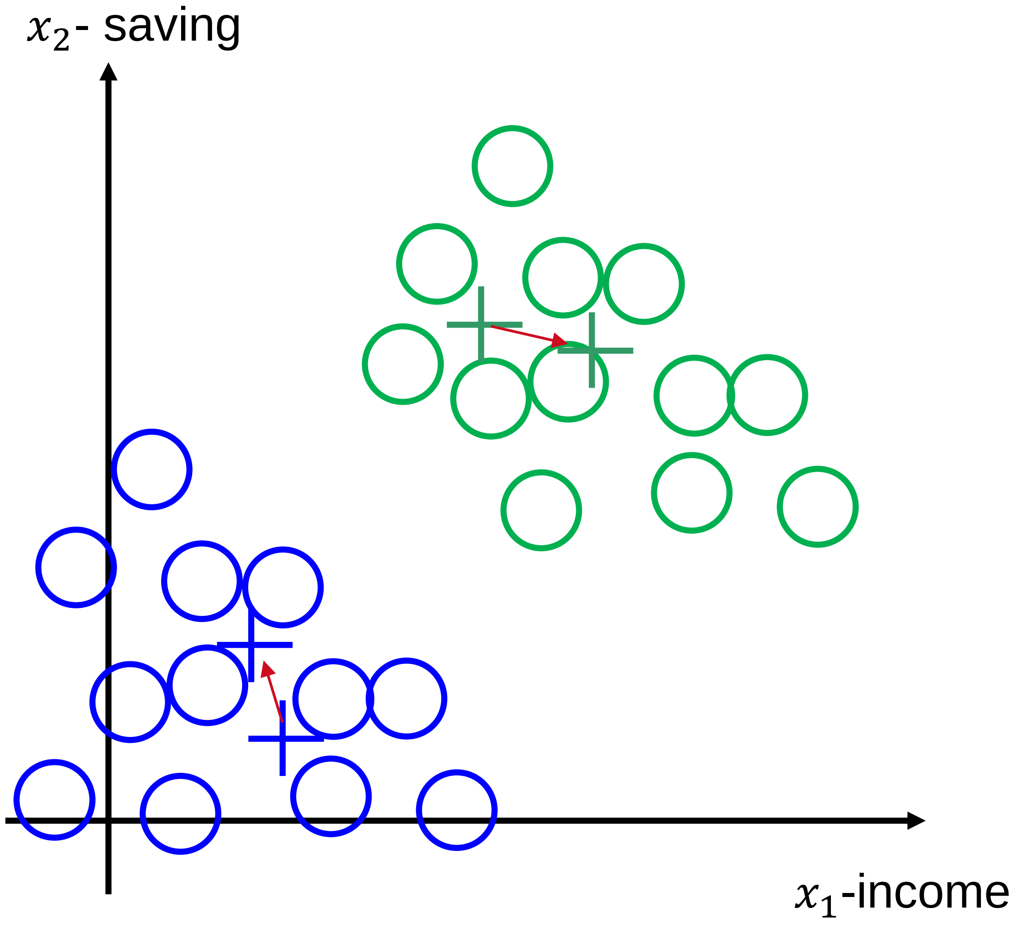
Loop 2

New Centroids' locations.



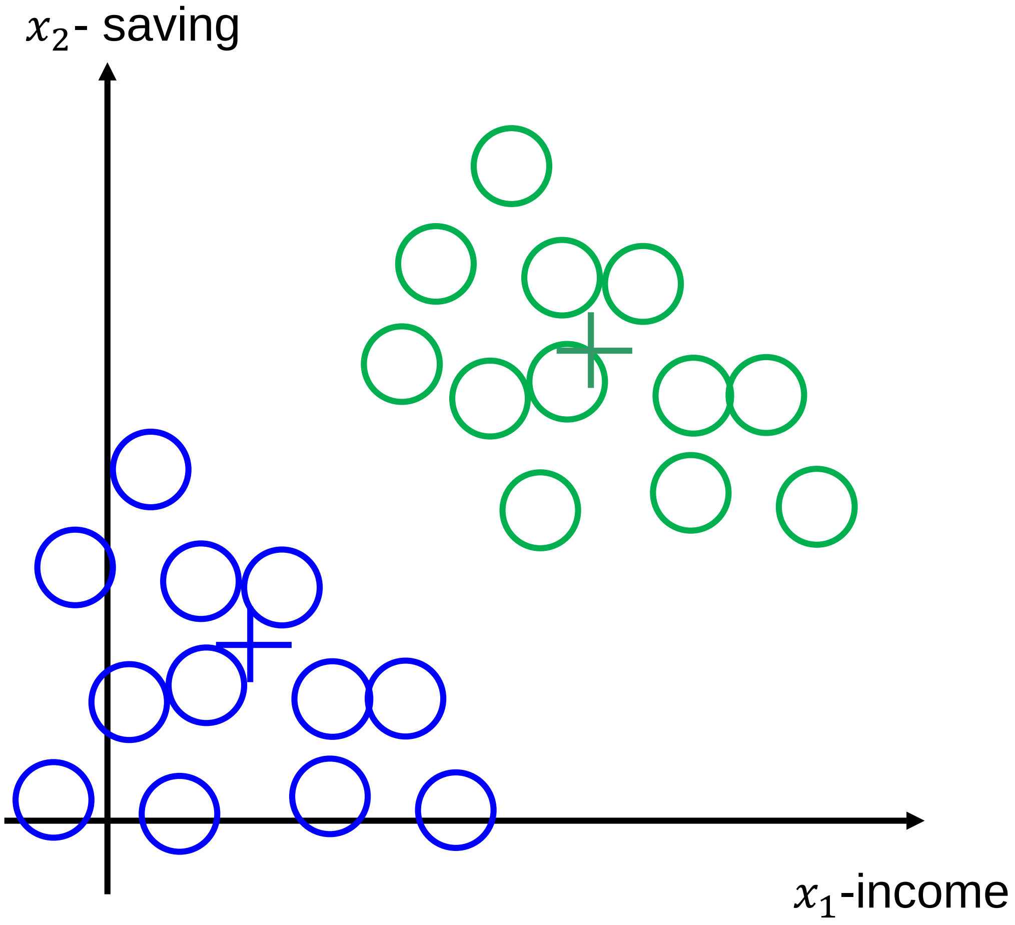
Loop 3

Cluster Assignment Step.



Loop 3

Centroid Move Step.



Loop 3

New Centroids' locations.

Keep running additional iterations of K-means until the cluster centroids will not change anymore and the clusters of the observations will not change any more.

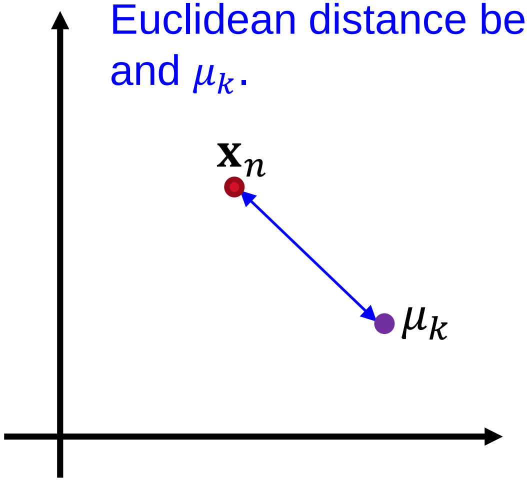
K-means Loss Function

C_n: the index of clusters (1,2,…,K) to which data point $\mathbf{x}_n$ is assigned (at the moment)
μ_k: cluster centroid k. Total number of clusters: K.
μ_{C_n}: cluster centroid to which data point $\mathbf{x}_n$ was assigned.

Suppose $\mathbf{x}_n$ is allocated to cluster 2 ($C_n = 2$), then $\mu_{C_n} = \mu_2$

Loss function (distortion): to be minimized

$$L(\boldsymbol{\beta}) = \sum_{n=1}^{N} \left\| x_n - \mu_{C_n} \right\|^2$$



Euclidean distance between $\mathbf{x}_n$ and μ_k.

$\boldsymbol{\beta}$: index of clusters and cluster centroids

K-means Algorithm

Randomly initialize K cluster centroids, $\mu_1, \mu_2, \mu_3, \dots, \mu_K$

Loop the two step process until converge:

Cluster Assignment Step.

for *n* in *1:N*

C_n: assign the observations $\mathbf{x}_n$ to the closest centroid μ_k.

In this step, we fix $\mu_1, \mu_2, \mu_3, \dots, \mu_K$, and aim to minimize the loss function $\mathrm{L}(\boldsymbol{\beta})$ with respect to $C_1, C_2, C_3, \dots, C_N$.

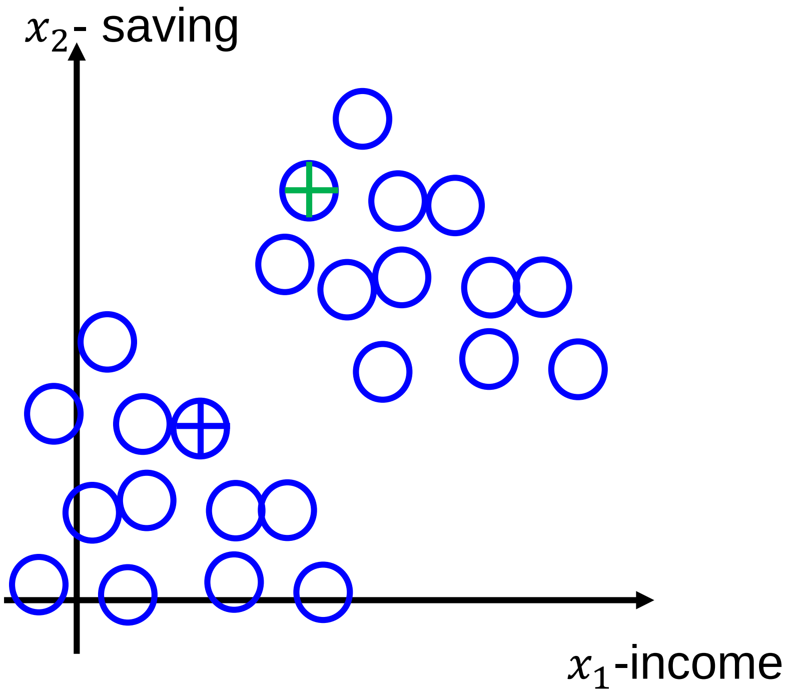
Centroid Move Step.

for *k* in 1:K

μ_k: mean of the data points assigned to cluster *k* => as the new cluster centroid

In this step, we fix $C_1, C_2, C_3, \dots, C_N$, and aim to minimize the loss function $\mathrm{L}(\boldsymbol{\beta})$ with respect to $\mu_1, \mu_2, \mu_3, \dots, \mu_K$.

How to initialize the K-means?



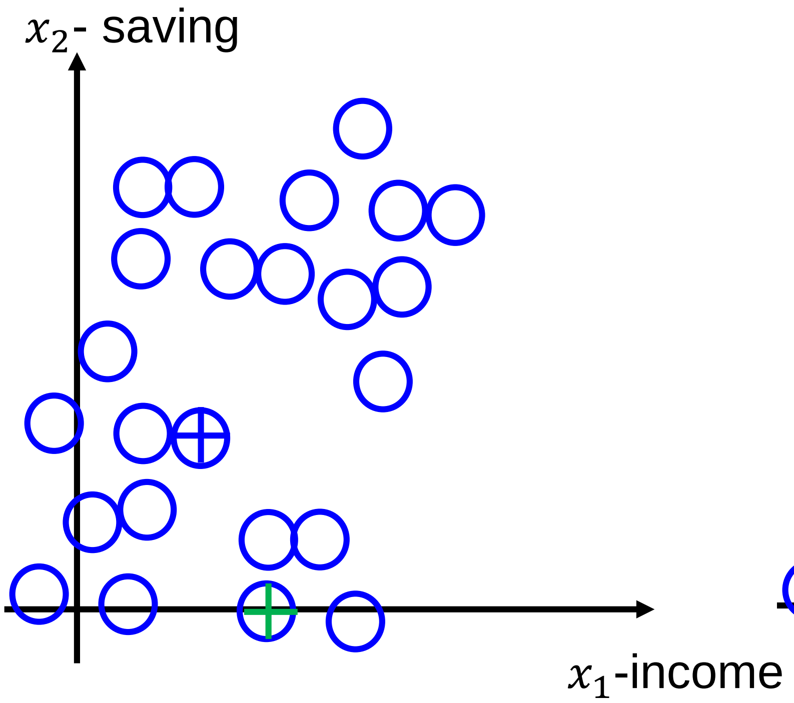
Random initialization:

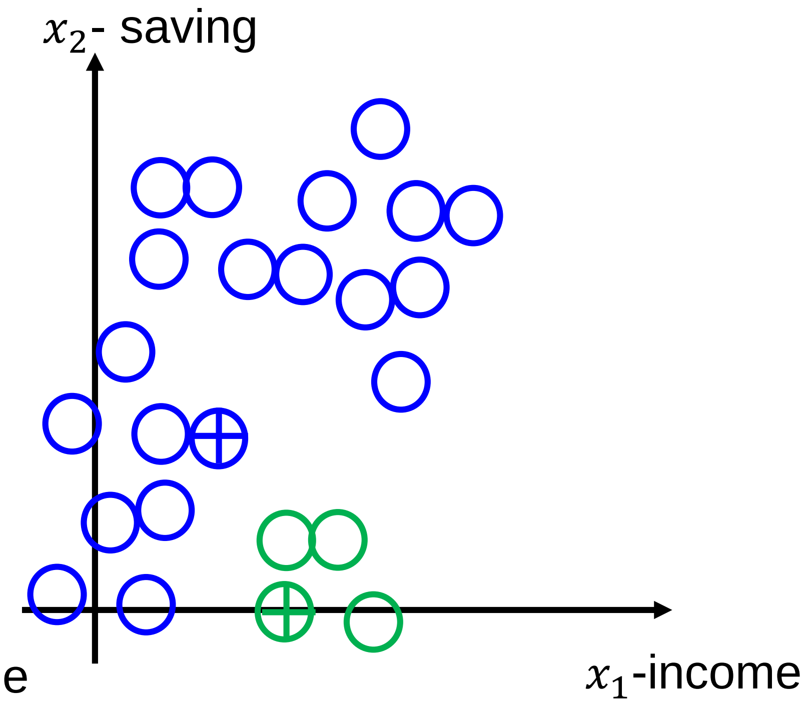
K<N: the number of cluster centroids, K, set to be less than the number of training examples m

Randomly select K training examples

Set $\mu_1, \mu_2, \mu_3, \dots, \mu_K$ equal to these K observed training data points. For example: $\mu_1 = \mathbf{x}_i, \mu_2 = \mathbf{x}_j$

What if the random initialization gives us the following starting centroids. It is possible the algorithm falls into the **local optima**.





Summary

- A set of data $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \subset \mathbb{R}^d$ and the targeted number of clusters K
- The k-Means objective and its optimisation problem

$$\min_{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K, C_1, \dots, C_N} L(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K, C_1, \dots, C_N) = \sum_{n=1}^{N} \left\| \mathbf{x}_n - \boldsymbol{\mu}_{C_n} \right\|^2$$

- where $\boldsymbol{\mu}_k \in \mathbb{R}^d$ $(k = 1,2, \dots, K)$ are centroids, and C_n are discrete indicator variable taking values $1, 2, \dots, K$
- An optimal problem over two sets of variables $\boldsymbol{\mu}$ and $\mathbf{C}$, continuous and discrete
- Alternating way to solve it:
  - ✓ Step 1: Updating $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K)$ while $\mathbf{C} = \{C_1, \dots, C_N\}$ is fixed
  - ✓ Step 2: Updating each C_n while $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_K)$ is fixed (that is, may change $\mathbf{x}_n's$ cluster)

Extensions

- Change from the squared Euclidean distance to Euclidean distance

$$\min_{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\ldots,\boldsymbol{\mu}_K,C_1,\ldots,C_N} L(\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\ldots,\boldsymbol{\mu}_K,C_1,\ldots,C_N)=\sum_{n=1}^{N}\left\|\mathbf{x}_n-\boldsymbol{\mu}_{C_n}\right\|$$

- Use other distances

$$\min_{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\ldots,\boldsymbol{\mu}_K,C_1,\ldots,C_N} L(\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\ldots,\boldsymbol{\mu}_K,C_1,\ldots,C_N)=\sum_{n=1}^{N}\mathrm{dist}(\mathbf{x}_n,\boldsymbol{\mu}_{C_n})^2$$

With or without the square

- Mixture of Gaussians:
  - ✓ We are not firmly assigning a datum $\mathbf{x}_n$ to one of K clusters, but calculating the probability of being each cluster, $\pi_{n1}, \pi_{n2}, \ldots, \pi_{nK}$

How to deal with local optima problem?

Repeat the random initialization for *T*, e.g. 100, times

For each initialization:

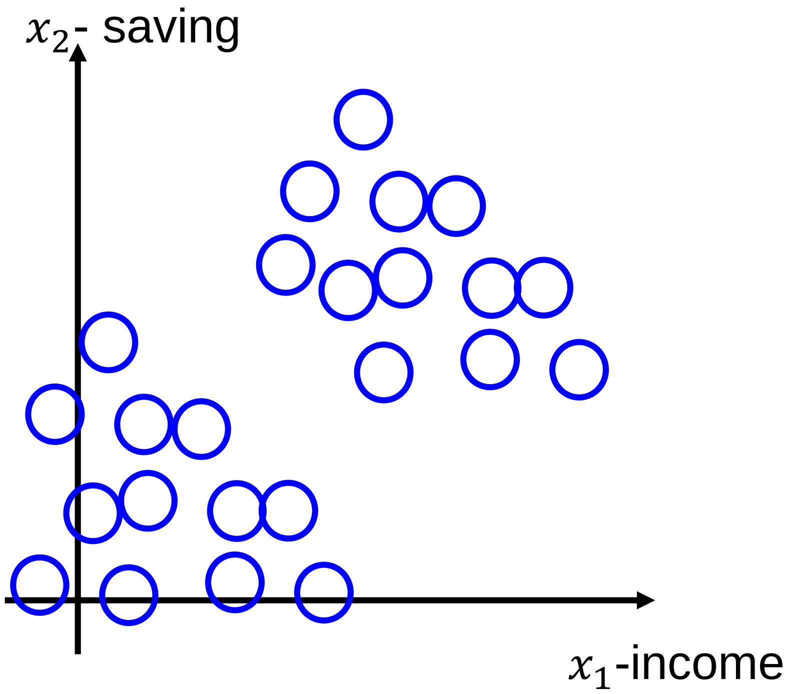
- Run the k-means algorithm, and record loss function $\mathrm{L}(\boldsymbol{\beta})$ values

- Pick the initialization that produces the lowest $\mathrm{L}(\boldsymbol{\beta})$

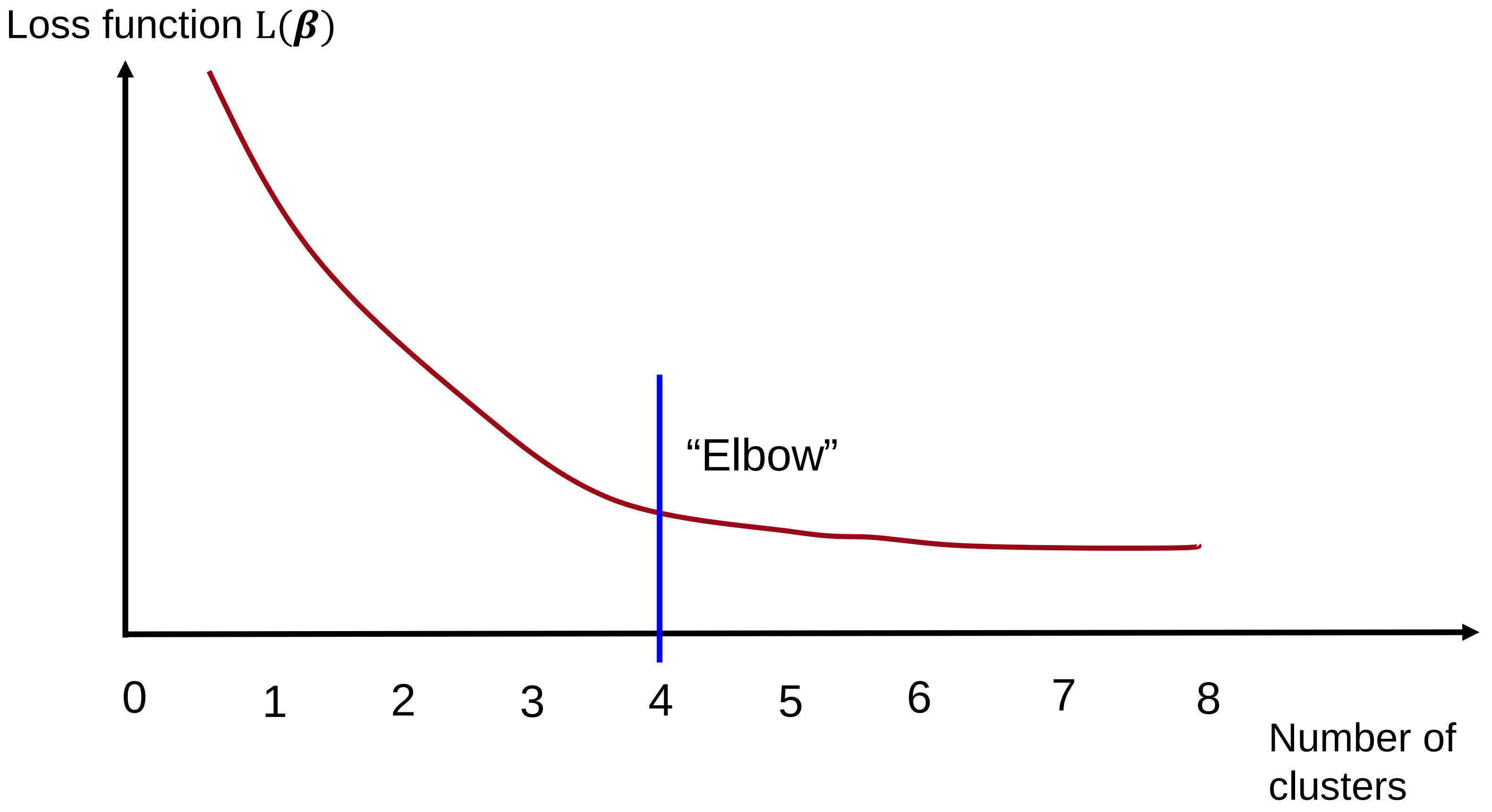
How to choose K?

No agreed way

Consider your application based on the real world scenario, e.g. clustering the female and male customers (*K*=2 is good); clustering the size of shoes (*K*=2 is not a good choice).



“Elbow” Method



Python Example

(Lecture04_Example02.py)